Quiz/Homework Assignment #2
CSE326
Homework due date:
Wednesday, January 19th, beginning of class

January 20, 2000

This quiz is homework assignment #2. Any question you answered correctly in class were considered correct for the purposes of the assignment. Any question you did not answer or which you answered incorrectly was a short answer question for the homework assignment due in class on Wednesday.

In *italics* are instructions on how to answer each question for homework. The correct letter answer and an explanation of that answer is listed after each question.

Good luck!

1. *Explain briefly little-o. Reduce each expression to simplest form. List most descriptive asymptotic relationship between each pair.*

   Which of the following are listed in strictly increasing asymptotic order (i.e., the left function is asymptotically less than, or little-o, of the right function)?

   (a) 1, 2
   (b) $\log^2 n$, $n \log n + \log n$
   (c) n^2, $n\sqrt{n}$
   (d) $2^{\log n}$, $5n$
   (e) 3^n, n^3

   Correct answer(s): **(b)**
   f_1 is little-o of f_2 if and only if f_1 is asymptotically upper bounded by f_2 *and* f_1 is not $\Theta(f_2)$ (i.e., they're not tight bounds for each other).
   The left functions of (a), (c), (d), and (e) are not little-o of their right functions:

   - $1 = \Theta(2)$
   - $n^2 = \omega(n\sqrt{n})$
   - $2^{\log n} = n = \Theta(5n)$
   - $3^n = \omega(n^3)$

   However, $n \log n + \log n = \Theta(n \log n)$ and $\log^2 n = o(n \log n)$. So, $\log^2 n = o(n \log n + \log n)$.

2. *Derive bound for recurrence (as we did in class).*

   Which of the following is an asymptotically-tight (i.e. big-Θ) bound for the recurrence: $T(n) = T(n/2) + 1$?

(a) $\Theta(n^2)$
(b) $\Theta(n)$
(c) $\Theta(\log n)$
(d) $\Theta(n \log n)$
(e) $\Theta(1)$

Correct answer(s): **(c)**

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 1 + 1 \\ &= T(n/8) + 1 + 1 + 1 \\ &= T(\frac{n}{2^k}) + k \end{aligned}$$

So, for $2^k = n$ or $k = \log n$ and assuming $T(1) = \Theta(1)$:

$$\begin{aligned} T(n) &= T(\frac{n}{n}) + \log n \\ &= T(1) + \log n \\ &= \Theta(\log n) \end{aligned}$$

3. *Note time for each statement. Write recurrence describing time taken by function. Derive bound for recurrence.*

The following function is designed to make a computer user say that they see an albatross. It does this by asking them again and again until they go crazy and say that they do see one:

```
int Albatross(int n)
  If n=1 Then
    Return
  End If

  For i=1 to n
    Print ''Do you see the albatross now?''
  End For

  Albatross(n/2)
```

Which of the following is an asymptotically-tight bound on the number of times the user would be asked about the albatross:

(a) $\Theta(n^2)$
(b) $\Theta(n)$
(c) $\Theta(\log n)$
(d) $\Theta(n \log n)$
(e) $\Theta(1)$

Correct answer(s): **(b)**

```
int Albatross(int n)
    If n=1 Then                                  O(1)
        Return                                   O(1)
    End If                                       O(1)

    For i=1 to n                                 n times through loop
        Print "Do you see the albatross now?"    O(1)
    End For                                      O(1)

    Albatross(n/2)                               T(n/2)
```

Note that $T(1) = \Theta(1)$ since the function will stop at the `Return` if $n = 1$.
Total time then is:

$$\begin{aligned} T(n) &= c_1 + c_2 n + T(n/2) \\ &= T(n/4) + c_2 n + c_2 n/2 + c_1 + c_1 \\ &= T(n/8) + c_2 n + c_2 n/2 + c_2 n/4 + c_1 + c_1 + c_1 \\ &= T(\frac{n}{2^k}) + \sum_{i=0}^{k-1} \frac{c_1 n}{2^i} + \sum_{i=1}^{k} c_1 \end{aligned}$$

So, for $2^k = n$ or $k = \log n$ and given $T(1) = \Theta(1)$:

$$\begin{aligned} T(n) &= T(\frac{n}{n}) + \sum_{i=0}^{\log n - 1} \frac{c_2 n}{2^i} + \sum_{i=1}^{\log n} c_1 \\ T(n) &= T(1) + n * \sum_{i=0}^{\log n - 1} \frac{1}{2^i} + c_1 \log n \end{aligned}$$

Here, I appeal to the book which tells us that $\sum_{i=1}^{\infty} \frac{1}{2^i} = \frac{1}{1-1/2} = 2$ and we know that $\sum_{i=1}^{1} \frac{1}{2^i} = 1/2$. So, every value of this sum is between 1/2 and 2 and it is $\Theta(1)$ regardless of the maximum value of i! (Our sum is actually a touch larger, but still constant.) Back to our regularly scheduled analysis:

$$\begin{aligned} T(n) &= T(1) + n * \Theta(1) + c_1 \log n \\ &= \Theta(1) + \Theta(n) + \Theta(\log n) = \Theta(n) \end{aligned}$$

4. *Derive correct equation (step by step).*

   Consider the following recurrence relation:

   $T(n) = 2T(n/3) + \sqrt{n}$

   Suppose we expand this equation k recursive times as we did in class (assume that the recurrence stated above is correct for $k = 1$). Then, in general we would have which of the following?

(a) $T(n) = kn^{\log_3 2}$

(b) $T(n) = 2^k T(n/3^k) + \sum_{i=1}^{k} 2^{i-1}\sqrt{n/3^{i-1}}$

(c) $T(n) = 2kT(n/3^k) + \sqrt{n/k}$

(d) $T(n) = 2kT(n/3k) + \sqrt{n+1-k}$

(e) $T(n) = k/2T(n/3k) + k\sqrt{k}$

Correct answer(s): **(b)**

$$\begin{aligned}
T(n) &= 2T(n/3) + \sqrt{n} \\
&= 2(2T(n/9) + \sqrt{n/3}) + \sqrt{n} \\
&= 4T(n/9) + 2\sqrt{n/3} + \sqrt{n} \\
&= 4(2T(n/27) + \sqrt{n/9}) + 2\sqrt{n/3} + \sqrt{n} \\
&= 8T(n/27) + 4\sqrt{n/9} + 2\sqrt{n/3} + \sqrt{n} \\
&= 2^k T(n/3^k) + \sum_{i=1}^{k} 2^{i-1}\sqrt{n/3^{i-1}}
\end{aligned}$$

5. *Justify correct answer (how do we know it?). Give correct form for (b), (d), and (e) (e.g., what is a good bound for the maximum number of leaves in the tree?). Briefly explain why (c) is not true.*

A certain tree has a depth of d, a maximum branching factor of b, and n nodes total. Which of the following must be true of the tree?

(a) The tree has $n - 1$ edges.

(b) The tree has no more than 2^d leaves.

(c) The tree's smallest element is stored at its leftmost leaf.

(d) n is no larger than d^b

(e) Some node in the tree has d children.

Correct answer(s): **(a)**

- Every node in the tree has exactly one parent (connected to it by an edge) except for the root. No other edges exist (or they would connect to a new node in which case the edge should already have been counted when we found that node's parent). Therefore, there are $n - 1$ child-to-parent edges and $n - 1$ edges total.
- The tree has no more than b^d leaves (that's the number of leaves a complete tree would have).
- There is implicit ordering on a binary tree. Binary search trees have the property that the tree's smallest element is stored at its leftmost leaf, but a general tree may have its smallest element in any node.
- The tree has no more than $\sum_{i=0}^{d} b^i = \frac{b^{d+1}-1}{b-1}$ nodes, summing the levels of a complete tree (the sum is solved on page 4 of the book). So, $n \le \frac{b^{d+1}-1}{b-1}$.

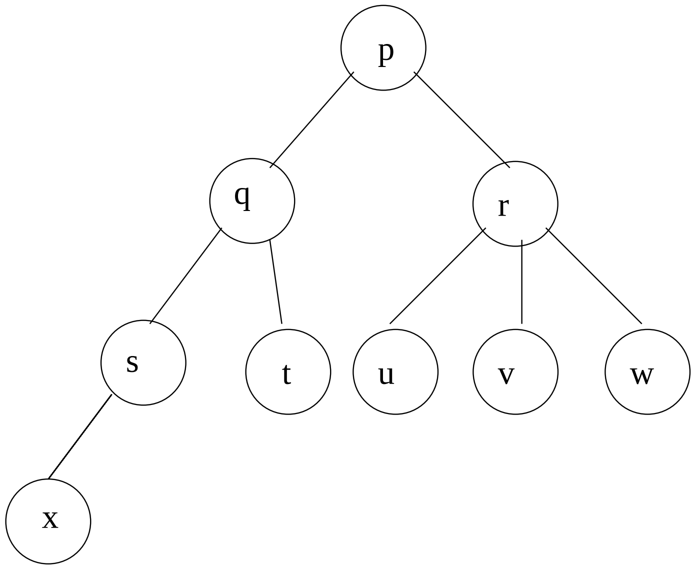

Figure 1: A tree with labelled nodes. Use this tree to answer questions 6-7.

6. *Name the nodes in each relationship (e.g., list nodes which are children of* **r***).*

   Which of the following is **not** equal to 3?

   (a) the number of children of **r**
   (b) the height of **p**
   (c) the number of ancestors of **t**
   (d) the depth of **x**
   (e) the depth of **u**

   Correct answer(s): **(e)**

   (a) **r**'s children are **u**, **v**, and **w**.
   (b) The height of **p** is 3 because the three edges on the path **p** $\Rightarrow$ **q** $\Rightarrow$ **s** $\Rightarrow$ **x** form the longest root to leaf path in the tree.
   (c) **t**'s ancestors are **t**, **q**, and **p**.
   (d) The depth of **x** is 3 because the three edges on the path **p** $\Rightarrow$ **q** $\Rightarrow$ **s** $\Rightarrow$ **x** form the path from the root to **x**.
   (e) The depth of **u** is 2 because the two edges on the path **p** $\Rightarrow$ **r** $\Rightarrow$ **u** form the path from the root to **u**.

7. *Describe true relationship of each node in the answers (just one word answers are fine).*

   Which best describes the siblings of **u**?

   (a) **s**, **t**, **v**, **w**
   (b) **v**, **w**
   (c) **p**, **r**
   (d) **u**, **x**

(e) **s**, **t**, **u**, **v**, **w**

Correct answer(s): **(b)**

- **s** is **u**'s grandparent's grandchild. Essentially a cousin.
- **t** is in the same relationship to **u** as **s**.
- **v** is **u**'s sibling.
- **w** is **u**'s sibling.
- **p** is **u**'s grandparent (and ancestor).
- **r** is **u**'s parent (and ancestor).
- **u** *is* **u** (and so is **u**'s ancestor and descendant).
- **x** is **u**'s grandparent's great-grandchild. Essentially a cousin.

8. *Justify answer (why is it a good description), explain why (c) is a poor description, and explain how Priority Queues are similar to Stacks and Queues.*

The Queue ADT can be briefly described as FIFO (First In First Out) and the Stack ADT as LIFO (Last In First Out). Which of these best describes the Priority Queue ADT?

(a) Last In Last Out

(b) Always Best Out

(c) Never Highest Out

(d) Garbage In Garbage Out

(e) None of these; the Priority Queue ADT is totally unlike the Stack and Queue ADTs

Correct answer(s): **(b)**

Always Best Out is a good description of a Priority Queue because the supported operation which removes nodes (deleteMin) always deletes the one with the best (lowest) priority.

Never Highest Out is not a good description because a Priority Queue of size one will return its highest priority element when its deleteMin is called. Another acceptable reason would be that some priority queues implement deleteMax rather than deleteMin without loss of generality (they just define "best" differently).

Finally, Priority Queues are like Stacks and Queues in that all three represent a collection of homogenous (same-typed) elements with two fundamental operations: one to put new elements in and one to remove existing elements. Moreover, the order of removal is well-defined (although what exactly the definition is is where these ADTs differ).